REMARKS

Applicant has amended Figure 6 to correct the flow from item 28 and, in particular, the YES branch leading to item 30. See attached the replacement sheet.

The Examiner has rejected Claims 1-2, 4-5, 7-8, 12-14, 16-17, 19-20, 24-26, 28-29, 31-32, and 36 under 35 U.S.C. 103(a) as being unpatentable over Yann (U.S. Publication No. 2002/0078368), in view of Risch (U.S. Patent No. 5,471,629), in further view of Chambers (U.S. Patent No. 5,398,196). Applicant respectfully disagrees with such rejection, especially in view of the amendments made hereinabove to the independent claims. Specifically, applicant has amended the independent claims to at least substantially include the subject matter of former dependent Claim 8 et al.

With respect to the independent claims, the Examiner has relied on the following excerpts from Yann and Risch to make a prior art showing of applicant's claimed technique "wherein said analysis logic includes a dependence table indicating dependency between state variables within said computer and loaded variable values, and for each program instruction said analysis logic makes a determination as to which state variables are read and written by that program instruction and for each loaded variable value within said dependence table if any state variable read by that program instruction is marked as dependent upon said loaded variable value, then all state variables written by that program instruction are marked as dependent upon said loaded variable value with previous dependencies being cleared" (see this or similar, but not necessarily identical language in the independent claims).

"[0030] The present disclosure provides methods of detecting a polymorphic viral code at earlier stages of emulation. The operand and operator values of instructions emulated may be monitored. Special consideration may be given to unused values and those instances in which the operand/operator value is calculated without being defined. A plurality of these instances found during emulation may contribute to a determination that a polymorphic viral code is present. Once the suspect instances are identified, there is a higher probability that the code is viral or used by a virus decryptor. The emulator then is allowed to emulate a larger amount of instructions. Thus, emulation time is spent only on emulating code which has a high probability of containing a virus." (Yann, Paragraph 0030)

'[0034] The conditions of the register/flag triggering an improper register usage state are shown in the state diagram in FIG. 3. **During emulation of the program, a register/flag may be in one of the following three states: (a) undefined; (b) set; and (c) used. If a register/flag is loaded with an unknown value, its state is "undefined." Once the value that is loaded into a register/flag is known, the register/flag is in a "set" state. If a value in the register/flag is employed by a CPU instruction, the register/flag is in a "used" state.**' (Yann, Paragraph 0034 - emphasis added)

"**A single Function Dependence table is used to correlate extensional functions with intensional functions.** As previously indicated, an intensional function accesses an attribute value by computation or the like based on other attribute values. A change in one of these other values may result in a change in the value accessed by that intensional function. Accordingly, if an attribute which is accessed by an intensional function is being monitored, any change in any of the values which are used in accessing the monitored attribute must be detected in order to determine whether the monitored value has changed. The Function Dependence table provides this correlation.

Accordingly, during monitor definition, if the function which accesses the attribute to be monitored is an intensional function, that function is listed in the Function Dependence table together with each extensional function which could change any of the values which are utilized by that intensional function in accessing the monitored attribute. Later, when Check Monitors is called, the extensional function update calls as listed in the Function Change table are compared with the listings in the Function Dependence table. In this way, the system detects update function calls which may have resulted in changes to monitored attributes (block 302)." (Risch, Col. 10, lines 28-51 - emphasis added).

In addition to the above excerpts, the Examiner argued that "Yann also discloses the limitation of part c with the exception that Yann is silent as to the monitored data being stored in a table. Risch teaches the well-known idea that data may be stored in a table."

Applicant respectfully asserts that the combination of excerpts from Yann and Risch fails to disclose all of applicant's claimed techniques. Specifically, Risch discloses "[a] single Function Dependence table [that] is <u>used to correlate</u> <u>extensional functions</u> with <u>intensional functions</u>" (emphasis added). Further, Yann teaches that "[d]uring emulation of the program, a <u>register/flag may be in one of the following three states</u>: (a) <u>undefined</u>; (b) <u>set</u>; and (c) <u>used</u>" (emphasis added). Yann continues to teach that register/flag states are "undefined" when "loaded with an unknown value," "set" when "the value that is loaded into a register/flag is known," and "used" when the "value in the register/flag is employed by a CPU instruction." However, combining Yann's register/flag states with Risch's "Function Dependence table" still fails to rise to the level of specificity of applicant's claimed technique.

Specifically, the combined excerpts simply fail to disclose a technique "wherein said analysis logic includes a dependence table <u>indicating dependency between state variables</u> within said computer and loaded variable values" (emphasis added), as claimed by applicant. In addition, the combined excerpts fail to disclose a technique where "for each loaded variable value within said dependence table if <u>any state variable read</u> by that program instruction is <u>marked as dependent upon said loaded variable value</u>, then <u>all state variables written</u> by that program instruction are <u>marked as dependent upon said loaded variable value</u> with previous dependencies being cleared" (emphasis added), as claimed by applicant.

To establish a *prima facie* case of obviousness, three basic criteria must be met. First, there must be some suggestion or motivation, either in the references themselves or in the knowledge generally available to one of ordinary skill in the art, to modify the reference or to combine reference teachings. Second, there must be a reasonable expectation of success. Finally, the prior art reference (or references when combined) must teach or suggest all the claim limitations. The teaching or suggestion to make the claimed combination and the reasonable expectation of success must both be found in the prior art and not based on applicant's disclosure. *In re Vaeck*, 947 F.2d 488, 20 USPQ2d 1438 (Fed.Cir.1991).

Applicant respectfully asserts that at least the third element of the *prima facie* case of obviousness has not been met, since the prior art references, when combined, fail to teach or suggest <u>all</u> of the claim limitations, as noted above. Nevertheless, despite such paramount deficiencies and in the spirit of expediting the prosecution of the present application, applicant has amended the independent claims to incorporate the subject matter of Claim 8 et al.

With respect to the subject matter of former Claim 8 et al. (now at least substantially incorporated into the independent claims), the Examiner has relied on the following excerpts from the Yann reference to make a prior art showing of applicant's claimed technique "wherein said analysis logic includes an initialisation table indicating which state variables have been initialised." Applicant has provided an additional excerpt from Yann to provide additional context on item 24 of Figure 2.

> '[0033] The operational code analyzer extracts information about any operands and/or operators involved in the emulated execution of the instruction along with the state of CPU registers and flags. **The operational code analyzer passes the gathered information to the heuristic processor. The heuristic processor collects and maintains data corresponding to each register/flag used by the emulated instruction.** The heuristic processor may maintain counters 26 corresponding to respective registers/flags, with each counter keeping track of the number of times the corresponding register/flag triggered improper register usage state.
>
> [0034] The conditions of the register/flag triggering an improper register usage state are shown in the state diagram in FIG. 3. During emulation of the program, a register/flag may be in one of the following three states: (a) undefined; (b) set; and (c) used. If a register/flag is loaded with an unknown value, its state is "undefined." Once the value that is loaded into a register/flag is known, the register/flag is in a "set" state. If a value in the register/flag is employed by a CPU instruction, the register/flag is in a "used" state.' (Yann, Paragraphs 0033-0034 - emphasis added)

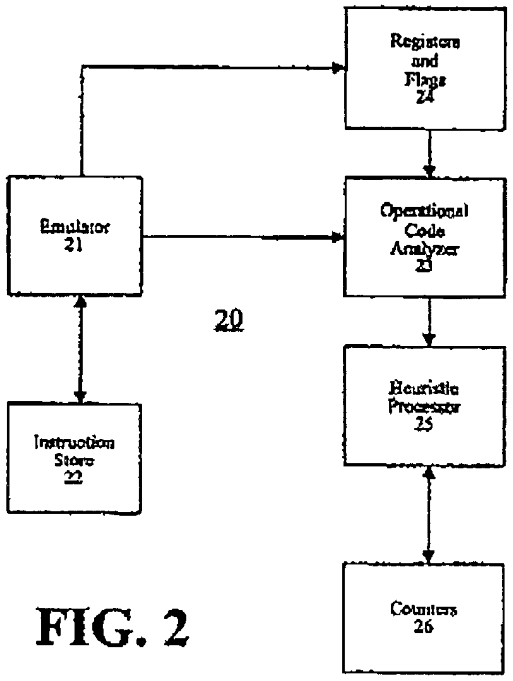

FIG. 2

(Yann, Figure 2, Item 24)

"States of registers and flags 24 are collected (step 14) and processed, along with the operands and operators, for the emulated instruction." (Yann, Paragraph 0031 excerpt - emphasis added)

Applicant respectfully asserts that the above excerpts from Yann relied upon by the Examiner disclose that the "[s]tates of registers and flags 24 are <u>collected</u>" (emphasis added). In addition, the excerpts from Yann disclose that "[t]he operational code analyzer passes the gathered information to the heuristic processor ... [which] ... <u>collects and maintains data corresponding to each register/flag</u> used by the emulated instruction" (emphasis added). Simply disclosing that states are collected and that the heuristic processor maintains data corresponding to each register/flag, however, simply fails to meet a technique "wherein said analysis logic includes an <u>initialisation table indicating</u> which <u>state variables have been initialised</u>" (emphasis added), as claimed by applicant.

Again, applicant respectfully asserts that at least the third element of the *prima facie* case of obviousness has not been met, since the prior art references, when combined, fail to teach or suggest <u>all</u> of the claim limitations. Still yet, for the purpose of further expediting the prosecution of the present application, applicant has also amended the independent claims to further distinguish applicant's claim language from the above references, as follows:

> "wherein said dependence table includes a column for each register within a processor, an external write, and a write to a flag value; and a plurality of rows with each row corresponding to a value loaded into said computer that influences a state of said computer;
>
> wherein said initialisation table includes a column for each register within said processor indicating whether each register is initialised" (see this or similar, but not necessarily identical language in the independent claims).

A notice of allowance or specific prior art showing of each of the foregoing claim elements, in combination with the remaining claimed features, is respectfully requested.

Still yet, applicant brings to the Examiner's attention the subject matter of new Claims 37-39 below, which are added for full consideration:

> "wherein if two different branch paths reach a common point, said different branch paths are treated as separate branch paths" (see Claim 37);
>
> "wherein upon reaching a conditional jump, said dependence table and said initialisation table are buffered; and upon reaching a target point of said conditional jump, a current version of said dependence table and said initialisation table and a previously buffered version of said dependence table and said initialisation table are merged" (see Claim 38); and
>
> "wherein upon reaching a loop, said dependence table and said initialisation table are buffered, where said dependence table and said initialisation table are merged during a next pass through said loop" (see Claim 39).

Again, a notice of allowance or specific prior art showing of each of the foregoing claim elements, in combination with the remaining claimed features, is respectfully requested.

Thus, all of the independent claims are deemed allowable. Moreover, the remaining dependent claims are further deemed allowable, in view of their dependence on such independent claims.

In the event a telephone conversation would expedite the prosecution of this application, the Examiner may reach the undersigned at (408) 505-5100. The Commissioner is authorized to charge any additional fees or credit any overpayment to Deposit Account No. 50-1351 (Order No. NAI1P460).

Respectfully submitted,
Zilka-Kotab, PC.

[illegible]

Kevin J. Zilka
Registration No. 41,429

P.O. Box 721120
San Jose, CA 95172-1120
408-505-5100